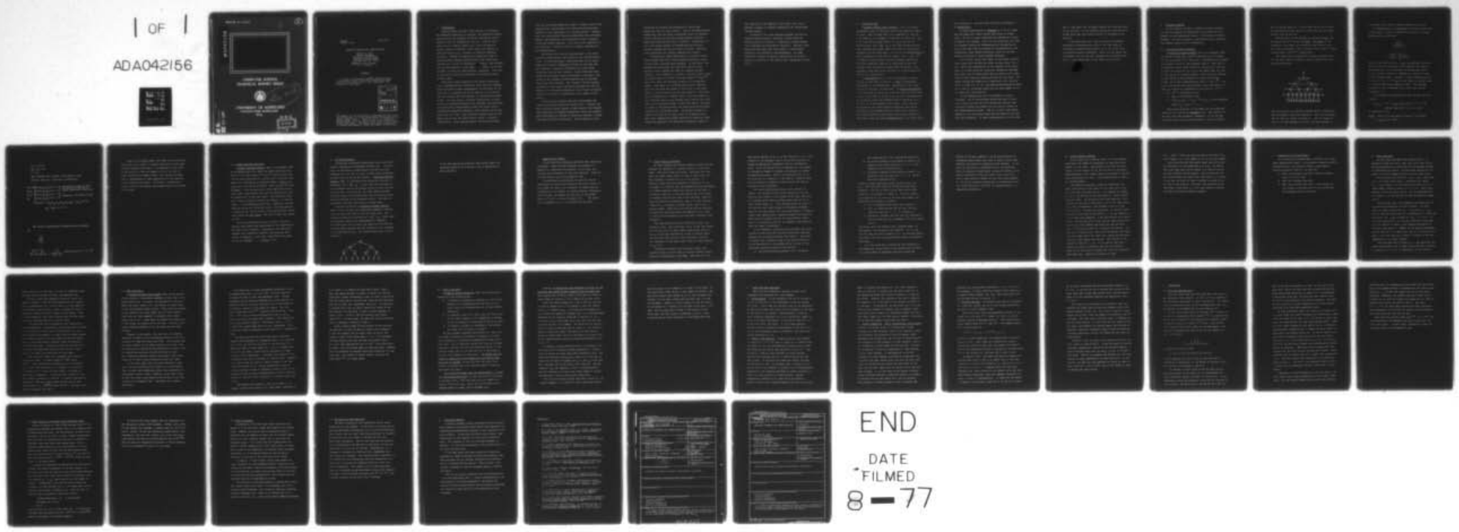
AD-A042 156 MARYLAND UNIV COLLEGE PARK COMPUTER SCIENCE CENTER F/G 9/4
CELLULAR PYRAMIDS FOR IMAGE ANALYSIS.(U)
MAY 77 C R DYER, A ROSENFELD AF-AFOSR-3271-77
UNCLASSIFIED TR-544 AFOSR-TR-77-0807 NL

1 OF 1
ADA042156



END
DATE
FILMED
8-77

AFOSR-TR- 77-0807

12

ADA042156

COMPUTER SCIENCE
TECHNICAL REPORT SERIES

See 1473

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

TR-544 May 1977
AFOSR-77-3271

CELLULAR PYRAMIDS FOR IMAGE ANALYSIS

Charles R. Dyer
Azriel Rosenfeld
University of Maryland
Computer Science Center
College Park, MD 20742

ABSTRACT

A class of multilayer bounded cellular arrays is defined for which many useful recognition tasks require time proportional to the logarithm of the diameter of the input.

The support of the Directorate of Mathematical and Information Sciences, U. S. Air Force Office of Scientific Research, under Grant AFOSR-77-3271, is gratefully acknolwedged, as is the help of Mrs. Shelly Rowe in preparing this paper. The authors also wish to thank David L. Milgram and Robert L. Kirby for helpful discussions.

1. Introduction

Image analysis involves a wide variety of techniques for measuring properties of pictures, and extracting objects from pictures, for purposes of pattern recognition and description (for an introduction, see [1]). Some of these tasks are of a general nature (e.g., local averaging or taking a Fourier transform), while others are specific to image processing (e.g., thinning or connected component counting). Methods of reducing the computation time of various general-purpose algorithms have been extensively studied; see [2] for an introduction. However, these methods have not been applied, to any appreciable extent, to the development of efficient image analysis techniques. Even less is known about efficient algorithms for performing image-specific tasks.

Many basic image recognition and analysis algorithms can be implemented very efficiently using parallel hardware. These algorithms operate independently on each point of the image and its neighbors and do not make use of any results that may already have been obtained at previously processed points. For this reason, "cellular" parallelism, in which similar or identical processors are uniformly interconnected, can perform many image analysis tasks much faster than a conventional sequential processor which examines the picture points one at a time. Each processor operates upon its own memory and has its own local control, which is globally synchronized. Since synchronization, uniform interconnection and distributed memory are fixed in advance, much of the overhead that characterizes less restricted multiprocessor computers is not applicable in a cellular machine. Furthermore, the restrictions facilitate algorithms which uniformly measure and agglomerate local properties using a balanced divide-and-conquer approach. Also, due to the regularity of its structure, a cellular array is a natural candidate for LSI implementation.

One problem with cellular architectures is the interprocessor communication time defined by the interconnection links. As mentioned above, algorithms which can be decomposed into mutually independent subtasks of the same form and complexity, requiring information only from nearby processors, are the most suitable. Under conditions where communication between distant processors is necessary, variations on the standard cellular approach may be desirable which lower this overhead. The "pyramid" cellular machine introduced in this paper configures processors so that the maximum of the distances between any two of them is logarithmically, not linearly, proportional to the number of processors.

Cellular array automata have been investigated previously as pattern recognition devices [3-6]. However, the use of cellular arrays imposes a lower bound on the inherent time complexity of language recognition, since any neighborhood interconnection network of processors requires a linear time lower bound for recognition. We can improve on this limitation by allowing the processors to occupy more dimensions than the input pattern. Such an extended-parallel machine may view the input at different levels of representation, and may also provide straightforward structural means for going from one level to another. Perceptrons [7] are a well-known example of extended parallelism which divide a computation into two distinct stages: measuring local properties and linearly combining the results.

A particularly simple modification of cellular arrays which extends them into a third dimension defines a class of multi-level cellular array automata. In particular, the pyramid data structures that have been used for image analysis [8-10] suggest the idea of a pyramid cellular acceptor. This acceptor is a stack of cellular arrays, where the bottom array is 2^r-by-2^r, the next lowest 2^{r-1}-by-2^{r-1}, and so on, until the top array consists of a single processor. Each layer is a cellular array, and in addition, each cell can sense the states of four cells in the layer below it, and of one cell in the layer above it. Thus each cell is connected to a "father" cell in the level above, to four "brother" cells in the current level, and to four "son" cells (in a two-by-two block) in the level below. The input is stored in the bottom array, the other cells being placed initially in a quiescent state. The pyramid accepts its input if the cell at its apex enters an accepting state. This novel computational model improves the potential lower bound time complexity for nontrivial recognition tasks to the logarithm of the diameter of the input, with only a moderate increase in hardware complexity over conventional cellular arrays.

In Section 3 we define cellular pyramids (as well as certain simplifications), following a brief review of cellular arrays in Section 2. Some basic pyramid recognition algorithms are described in Section 4. Section 5 compares this new model with three others -- finite-state acceptors, tree acceptors, and perceptrons. We study additional language recognition capabilities of cellular pyramids in Section 6, and discuss their limitations in Section 7.

2. Cellular Arrays

A bounded cellular array acceptor (CA) is a finite, rectangular array of identical finite state machines (FSM's), or cells. Each of these cells is a quadruple $M = (Q_N, Q_T, \delta, A)$ where Q_N is a nonempty, finite set of states, $Q_T \subseteq Q_N$ is a finite set of input states, $A \subseteq Q_N$ is the set of accept states, and δ: $Q_N^5 \rightarrow Q_N$ is the state transition function, mapping the current states of M and its four nearest neighbors into M's next state. If M is to be nondeterministic then the mapping is into sets of states, i.e., δ: $Q_N^5 \rightarrow 2^{Q_N}$. In addition, there exists a special boundary state $\# \in Q_N$. The state transition function is restricted so that the boundary state can never be exited from or entered. Consequently, only those cells initially in a non-# state can ever be in a non-# state.

A configuration of a CA is an assignment of states from Q_N to each cell, in the CA. A step of computation corresponds to the simultaneous application of the state transition function δ at each cell. An input configuration is a configuration before the first step such that the state of the (i,j)th cell is in Q_T if all of (i,j)'s neighbors are in the array, and in the boundary state # otherwise (i.e., if (i,j) is on the border of the array). Due to the restrictions on the state transition function those cells initially in the boundary state # (which form a border of unit width around the non-# cells) must forever remain in that state; they are called boundary cells. All other cells are initially in some non-# state and will be referred to as retina cells.

An input configuration is accepted by a CA if at some step the upper-left corner retina cell enters an accept state. The set of input configurations accepted by a given CA defines its language. Smith [5] has shown that the class of languages accepted by nondeterministic (deterministic) CA's is the same class as accepted by nondeterministic (deterministic) array bounded acceptors, which are the two-dimensional analog of linear bounded acceptors.

For an m-by-n array, the number of time steps required before every cell can know the state of every other cell is m+n. Hence time proportional to the diameter of the input array is a lower bound for nontrivial recognition problems. Many tasks that at first glance seem to require $O(n^2)$ steps for an n-by-n array can actually be done in O(n) steps. Examples include the detection of connectivity [4], majority (more 1's than 0's) [5, 6], and packing(transforming an array of 0's and 1's to another array with the same number of 1's, but upper-left justified) [6].

In one dimension the CA is an acceptor of string languages. All of the previous definitions hold in this case, except that now the state transition function maps triples of states into states (or into sets of states, in the nondeterministic case). That is, a cell's next state depends on its own current state and the states of its left and right neighbors. An input configuration is of the form

#x#, x $\in Q_T^*$, where the ith symbol denotes the starting state of the ith cell. If the leftmost retina cell ever enters an accept state then the initial string x is accepted by the given CA.

It has been shown [11] that the class of languages accepted by nondeterministic CA's is the same class as accepted by nondeterministic LBA's (i.e., it is the class of context-sensitive languages). Smith [11] has shown that palindrome, periodic and Dyck languages are recognizable by CA's in time proportional to the length of the input.

3. Cellular Pyramids

In this section we define a pyramid cellular acceptor. In order to simplify the exposition of this model we first present its one-dimensional analog -- the triangle cellular acceptor. We also introduce a simplification which restricts the neighbor relation to be "bottom-up".

3.1 Triangle Cellular Acceptors

A triangle cellular acceptor (TA) is a triangular stack of one-dimensional CA's, where the bottom row is of length 2^r, the next lowest 2^{r-1}, and so on. Surrounding the triangle is a border of cells in the boundary state #. Each cell is an identical finite state machine $M = (Q_N, Q_T, \delta, A)$, where Q_N is again the state set, $Q_T \subseteq Q_N$ is the input state set, and $A \subseteq Q_N$ the set of accepting states. The transition function is now defined to map sextuples of states into states, $\delta: Q_N^6 \rightarrow Q_N$ (in the nondeterministic case, $\delta: Q_N^6 \rightarrow 2^{Q_N}$). As with the CA we have the usual boundary state restrictions in order to confine the computation to the size $2^r + 2^{r-1} + \ldots + 2^0 = 2^{r+1}-1$ triangular retina:

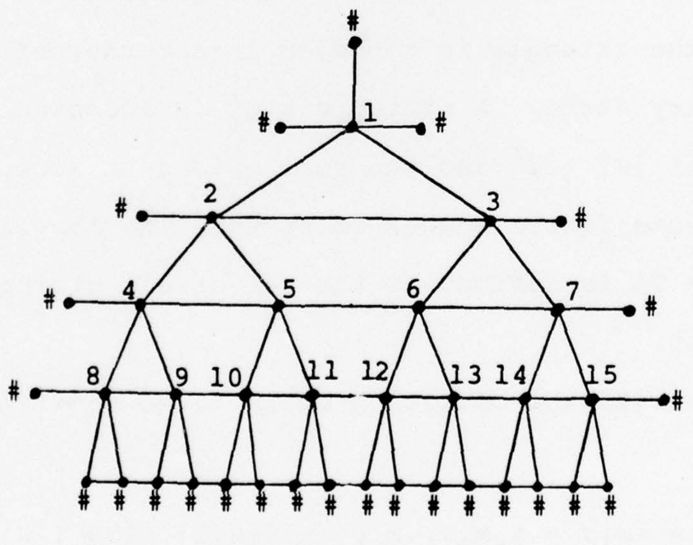
1) $\# \in Q_N$ is the boundary state

2) $\delta(q_{i_1}, q_{i_2}, q_{i_3}, \ldots, q_{i_6}) = \#$ iff $q_{i_1} = \#$ for arbitrary states $q_{i_j} \in Q_N, 2 \geq j \leq 6$.

Those cells which are in the domain of the transition function δ are called the neighbors of M. They consist of M's left and right neighbors ("brothers") in its own row, two "son" cells of M in the row below it, and one "father" cell in the row above it. If M is the ith cell in its row, its sons are cells 2i-1 and 2i in their row, and its father is cell $\lceil i/2 \rceil$ in its row.

The apex cell in the triangle is called the root, and the bottom row of 2^r cells, the base. The height of the triangle is the length of the shortest path from the root to a cell in the base. Thus a triangle with a base of length 2^r has height r and $2^{r+1}-1$ retina cells.

The figure below illustrates the graph representation of a TA with a base of length 8, where neighbor cells have been joined by arcs.



The retina cells are labelled (breadth-first) 1 through 15, and the boundary cells are labelled #. The root has label 1 and the base has labels 8 through 15. Cell 4's father is cell 2, its left brother is cell #, its right brother is

5, its left son is cell 8, and its right son is cell 9.

A configuration is an assignment of states to each cell in the retina. A step is defined as for the CA. An input configuration has the form:

```
                #
              # b #
             # b b #
           # b b b b #
                :
           # b b...b b #
        # q1q2 ...q2r-1q2r #
       #  # #  ...#  #  #
```

That is, the input string $\sigma = q_1 q_2 \ldots q_{2^r}$ defines the start states of cells in the base of the triangle, the rest of the retina cells are initialized to the quiescent state $b \in Q_T$, and the triangle is embedded in a border of cells in the boundary state. A string $\sigma \in Q_T^+$ is accepted by a TA of height r if $|\sigma| = 2^r$ and the root enters an accept state $q \in A$ after some finite number of steps. The language accepted by a TA is defined as the set of all strings accepted by it.

To illustrate the operation of TA's, we show that the language

$$L_{parity} = \{\sigma \mid \sigma = x_1 x_2 \ldots x_{2^k},\ x_i \in \{0,1\},\ 1 \le i \le 2^k$$
$$\text{and the number of 1's in } \sigma \text{ is even}\}$$

is accepted by a TA.

<u>Proof</u>: Define a TA with base of length 2^k as follows:

$M = (Q_N, Q_T, \delta, A)$, where

$Q_T = \{0,1,b\}$

$Q_N = Q_T \cup \{\#\}$

$A = \{0\}$

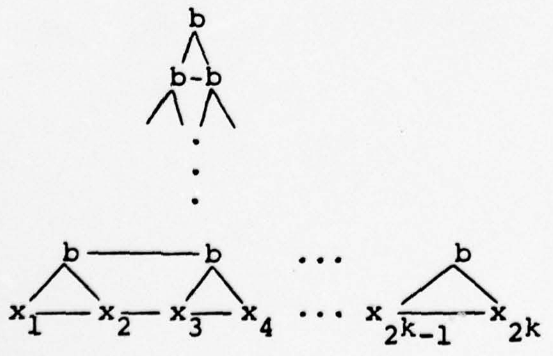
and δ(current cell, father, left brother, right brother, left son, right son) is defined by:

$$\begin{array}{l} \text{for all} \\ q_i, q_j, \\ q_k \text{ in} \\ Q_N \end{array} \left\{ \begin{array}{l} \left.\begin{array}{l}\delta(b,q_i,q_j,q_k,0,0) = 0\\ \delta(b,q_i,q_j,q_k,1,1) = 0\end{array}\right\} \begin{array}{l}\text{new parity of cell is even} \\ \text{if parity of left son is} \\ \text{equal to parity of right son}\end{array} \\ \left.\begin{array}{l}\delta(b,q_i,q_j,q_k,0,1) = 1\\ \delta(b,q_i,q_j,q_k,1,0) = 1\end{array}\right\} \text{otherwise, new parity is odd}\end{array}\right.$$

Otherwise, $\delta(q_{i_1},q_{i_2},q_{i_3},q_{i_4},q_{i_5},q_{i_6}) = q_{i_1}$ for all

$q_{i_j} \in Q_N,\ 1 \le j \le 6$

The initial configuration (boundary cells not shown) is:



where $x_i \in \{0,1\},\ 1 \le i \le 2^k$

Since a cell changes state only when its two son cells are in the 0 or 1 state, it follows that after k time steps the root changes from state b. By induction, if the state of the root is 0, then the number of 1's in the base is even, otherwise the number is odd. Thus the parity predicate is recognizable in time proportional to the logarithm of the length of the input. In contrast, recognition by either an LBA or CA requires time proportional to the length of the input.

3.2 Pyramid Cellular Acceptors

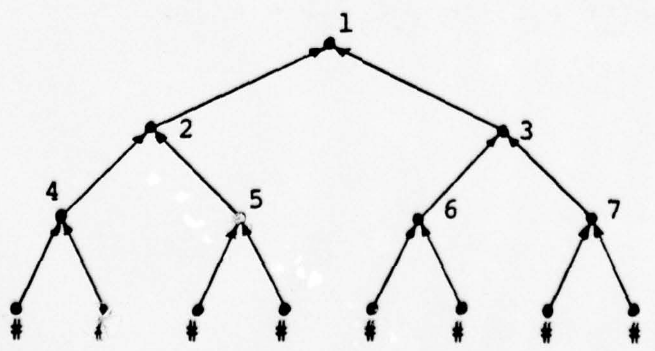
A pyramid cellular acceptor (PA) is a pyramidal stack of two-dimensional CA's, where the bottom array has size 2^r by 2^r, the next lowest 2^{r-1} by 2^{r-1}, and so forth, the (r+1)st layer consisting of a single cell, called the root. Each cell is defined as an identical FSM $M = (Q_N, Q_T, \delta, A)$. Q_N, Q_T and A are defined as before. Each cell now has nine neighbors -- four son cells in a two-by-two block in the level below, four brother cells in the current level, and one father cell in the level above. The transition function δ maps 10-tuples of states into states -- or sets of states, in the nondeterministic case. Otherwise, the conventions set up for the TA still hold. The input pattern is stored as the initial states of the bottom array, henceforth called the base array. The root is again the accepting cell.

Since the pyramid has only r+1 layers, a PA has a machine lower bound time proportional to the logarithm of the base array's diameter. Furthermore, the additional hardware cost required to achieve this potential time saving is moderate -- less than a third more cells than in the CA $(1+\frac{1}{4}+(\frac{1}{4})^2+\ldots = \frac{1}{1-(1/4)} = 4/3)$.

3.3 Bottom-up Machines

Alternative neighborhood definitions can be made which restrict information transmission through a PA. In particular, we now define a simplification in which the only neighbors of a cell are its sons, so that state information can move only one way up the pyramid. A bottom-up pyramid acceptor (BPA) is a PA whose state transition function is modified to be $\delta: Q_N^5 \rightarrow Q_N$. In this case the next state of a cell depends only on the current states of that cell and its four sons. As in the PA, the input pattern defines the start states of the base array, the other retina cells being initialized to the quiescent state b. The input is accepted if the root ever enters an accept state.

Similarly we define a bottom-up triangle acceptor (BTA) to be a TA with its neighborhood reduced to include only the given cell and its left and right sons. The transition function is defined as $\delta: Q_N^3 \rightarrow Q_N$. Under these restrictions a BTA can be represented by a complete directed binary tree in which the nodes are cells and the directed edges indicate that the source cell is a neighbor of the destination cell. The figure below shows a BTA of height 2:



In the next section we establish some results about the acceptance power of PA's and TA's, and in particular of BPA's and BTA's.

4. Capabilities of BPA's

In this section we describe some basic BPA recognition algorithms. These include detecting the presence or absence of a specified local pattern in the input, and counting the occurrences of such local patterns. These algorithms require only O(log diameter) time.

In describing the algorithms, we will not use the terminology of states and transition functions; rather, we will simply say that a cell transmits or outputs certain information to its father, or receives certain information from its sons. We assume that the size of the pyramid base is 2^n by 2^n, so that the height is n. The layers will be numbered 0,1,...,n, starting from the base.

4.1 Local Property Detection

We first consider the trivial problem of detecting the presence (or absence) of a specific value z in the input image. This can be done as follows: Each base cell compares its input image value to z, and if they match, the cell transmits a "match bit" (1) to its father. In addition, every base cell transmits a "flag bit" (1) to its father, to indicate that the matching has been performed. If a father receives a match bit from any of its sons, it transmits a match bit to its own father; and in any case, it transmits the flag bit that it has received. When the apex cell receives the flag bit, then if it also receives a match bit, it outputs a success signal, and if not, a failure signal. Note that if we did not use the flag bit, the apex cell could not detect the absence of z's, since it would not know whether or not the match information has reached it.

The time required for this algorithm is just (log diameter) steps. More precisely, if t_m is the time needed for the base cells to match their input values with z's, and t_r is the time required for a cell to receive match and flag bits from its sons (and to perform a logical "or" operation on the match bits), then the time required is $t_m + nt_r$.

Detecting arbitrary local patterns, rather than specific values, is best done by allowing sidewise transmission of information in the base. Each base cell can then decide whether or not it is the "head cell" (e.g., the leftmost of the uppermost cells) of the given pattern, by accepting appropriate information from its neighboring cells. Since the patterns are of bounded size, the amount of information needed is bounded, and presents no special transmission problems. When a cell discovers that it is the head cell of a pattern, it can take on a special value, and the presence or absence of those values can then be detected as just described.

For a BPA, detecting arbitrary local patterns is harder. The problem is that if the pattern is in a bad position, say exactly in the middle of the base, the cells at a bounded height above the base cannot see all of it, hence cannot detect it. [This problem would not arise if the cells on each layer looked at overlapping neighborhoods on the layer below; but this would require a linearly, rather than exponentially, tapering pyramid, the height of which would be proportional to the base diameter rather than to its logarithm, so that log diameter time operation would no longer be possible.]

In one dimension, arbitrary local patterns can be detected by a BTA using the following brute-force approach. Suppose that the desired pattern has length k. Each cell at height $\log k$ recieves a copy of the entire portion B of the base below it, and decides which (if any) of the following conditions holds for that portion.

a) The entire pattern is present in B. We denote this condition by S (for "successful detection").

b) An initial segment of the pattern of length i is present at the right end of B, $1 \leq i < k$. This condition will be denoted by S_i.

c) A terminal segment of the pattern of length j is present at the left end of B, $1 \leq j < k$. We denote this condition by S'_j.

If none of these conditions holds, we denote that fact by F. Note that more than one of them can hold simultaneously. The set of those that do hold can be represented by a bit pattern of length 2k. This bit pattern is transmitted to the cells at the next layer above, and these cells now operate as follows:

1) If S is received from either son, transmit S.

2) If S_i is received from the left son, and S'_{k-i} from the right son, $1 \leq i < k$, transmit S.

3) Otherwise, transmit the left son's S'_j information and the right son's S_i information (if none, transmit F).

Thus these cells too transmit only a bounded amount of information. The process is now repeated. It is clear that the apex cell will transmit S iff. the desired pattern is present, and that the entire process takes log diameter time steps.

It is not clear how to generalize this algorithm to two dimensions, since pieces of the pattern can now occur in a large number of different positions around the borders of the base segment B, and we must preserve the information about where they occur in order to match them properly with pieces in neighboring base segments. In any case, the algorithm is inelegant even in one dimension, since it requires a large amount of pattern matching to be done by the layer of cells at height $\log k$, which also need a large amount of memory to copy their base segments B. The use of sidewise transmission of information to detect local patterns in the base, as suggested above, is undoubtedly preferable.

4.2 Local Property Counting

We next show that a BTA can count, in 2 log diameter time steps, the number of occurrences of some particular symbol z in its input. Specifically, the apex cell will output, at time steps n to 2n, the n+1 bits in the binary representation of this number (which lies between 0 and 2^n), least significant bit first. The generalization to BPA's is straightforward.

The counting algorithm is defined inductively; we shall prove that at time steps k to 2k, each cell in the kth layer above the base (k=0,1,...,n) outputs the number of z's in the portion of the base below it, least significant bit first. To initialize this, each base cell (k=0) outputs, at the end of time step 0, a 1 if its input value is z, and a 0 otherwise. Now consider a cell C in the (k+1)st layer. By induction hypothesis, it receives from its sons at the ends of time steps k,...,2k the numbers of z's in their base segments. C can now function as a serial adder: At time step k+1 it sums the two least significant bits of its sons' numbers and stores the sum and carry bits. At the end of step k+1 it outputs the sum bit. At step k+2 it receives the next least significant bits from its sons, adds them to the carry bit, stores the resultant carry bit, and outputs the sum bit. This process is repeated at steps k+3,...,2k. At the end of step 2k+1, C outputs the sum bit resulting from the last addition step, and at step 2k+2 it outputs the carry bit resulting from that step. Clearly C's outputs at steps

$k+1,\ldots,2k+2 = 2(k+1)$ are just the bits of the sum of its sons' outputs, i.e., the number of z's in C's base segment.

Note that the base cells must output their bits only at time step 0 if this algorithm is to work. To insure this, the input data can be erased; or a flag bit can be set that inhibits further output. Once we know that the base cells produce no output after time step 0, it is easy to see that the cells in layer k produce no output after the end of time step 2k. The total time required by the algorithm is $2nt_a$, where t_a is the time needed to perform one step of the serial addition process.

5. Comparisons with Other Models

In this section we compare BPA's (or BTA's) with three other computational models -- finite-state acceptors (FSA's), frontier-to-root tree acceptors (FTA's), and diameter-limited perceptrons (DLP's). We show that

a) BTA's are stronger than FSA's, and can simulate them in log real time

b) BPA's are stronger than FTA's

c) BTA's are stronger than DLP's, if we assume certain bounds on the precision of the coefficients and threshold used by the DLP's.

5.1 BTA's and FSA's

We first show that BTA's can simulate FSA's. It suffices to show this for an FSA, call it M, that is one-way and deterministic, since such M's can simulate two-way or nondeterministic FSA's. Strings whose lengths are not powers of 2 can be padded with #'s at their right ends.

Let the state set of M be $S = \{s_1, \ldots, s_m\}$. Each base cell, say having input value v, constructs the vector $(t_1, \ldots, t_m)$, where t_i is the state that M goes into if it reads symbol v while in state s_i, $1 \leq i \leq m$. This vector defines a function from S into S. (It is understood that if M reads a # it can continue to move but does not change state.)

Each non-base cell C now computes the composition of the functions defined by its sons' vectors. In other words, if these vectors are $(u_1, \ldots, u_m)$ and $(v_1, \ldots, v_m)$ for the left and right sons of C, respectively, C constructs the vector $(w_1, \ldots, w_m)$ in which $w_i = v_{u_i}$, $1 \leq i \leq m$. It is easily seen that w_i is just the state that M would end in if it started in state s_i and scanned the portion of the base lying below C. Indeed, by the previous paragraph this is true if C is a base cell; and if it is true for C's sons, it is also true for C, since C's vector is just the composition of its sons' vectors.

Thus the apex cell's vector $(z_1, \ldots, z_m)$ specifies the state that M ends in if it scans the entire base, starting in state s_i, $1 \leq i \leq m$. In particular, if M's initial state gives rise in this way to a final or accepting state, the apex cell can accept the base, and otherwise not.

If $t(k)$ is the time required to look up a value in a table of length k, then the time required for this simulation is essentially $t(r) + nt(m)$, where r is the number of input symbols (gray levels). Here t includes the time required to transmit the sons' vectors to the father. This is O(log diameter) time, and is faster than the FSA itself, since that requires time 2^n to scan the base.

It is not clear how to generalize this construction to two dimensions. The problem is that M can enter or leave a given portion B of the base at many places (i.e., anywhere along B's perimeter), so that we can no longer specify M's behavior relative to B by a vector of bounded length. Thus it is an open question whether a BPA can simulate a two-dimensional FSA. (Simulation should be possible for a "one-way" two-dimensional FSA that can only do a raster scan of its input; but such FSA's are very weak.)

BTA's can also accept many languages that cannot be accepted by FSA's. A simple example is $a^N b^N$ (where $N=2^{n-1}$). The algorithm for accepting this is as follows: If a cell's sons are both a's, it outputs a; if both b's, it outputs b; if a and b (in that order), it outputs T; otherwise, it outputs F. If the apex cell outputs T, the BTA accepts; otherwise, it rejects. A similar construction shows that a BTA can accept langauqes that are not even context-free, e.g., $a^M b^M c^M d^M$ (where $M=2^{n-2}$). Thus BTA's are strictly stronger than FSA's.

5.2 BPA's and FTA's

A frontier-to-root tree acceptor (FTA) can be defined [12] as having a finite-state automaton at each node of the given rooted tree. Initially, the processors at the twig nodes read their inputs (i.e., the symbols at those nodes, if any) and generate output states. These states are sensed by the fathers of the twig nodes, who use them (together with their own input symbols) to generate their output states; and so on. A node does not compute its output until it has sensed the outputs of all its sons. The root node's output indicates whether or not the input tree has been accepted.

Suppose, in particular, that the tree is a complete quad-tree of depth n; this corresponds exactly to the tree structure of a BPA with base size $2^n x 2^n$. If we assume that the input symbols are all blanks except at the base, then the operation of the FTA can be simulated by a single "wave" of information passing up the BPA from base to apex. This simulation requires exactly log diameter time (where computing the BTA's state transitions takes unit time).

On the other hand, there are some recognition tasks that a BPA can perform that it cannot do in log diameter time; we shall show immediately below that recognition of palindromes is such a task. Evidently, such tasks cannot be done by an FTA, since whatever the FTA does at all must be done in log diameter time. Thus BPA's are stronger than FTA's.

For simplicity, we treat palindrome recognition in the one-dimensional case, i.e., by a BTA. We first show that it cannot be done in less than O(diameter) time. Indeed, acceptance by the apex cell depends only on the sequence of states of each of its sons, and there are m^t such sequences of length t, where m is the number of states. Now each half of the base has $m^{(2^n-1)}$ possible configurations. Thus if $t < 2^{n-1}$, two of these configurations, say B_1 and B_2, must give rise to the same state sequence. This means that if the apex accepts $B_1B_1^R$ in t time steps, where $t < 2^{n-1}$, it must also accept $B_2B_1^R$, which is not a palindrome. Thus no BTA can accept palindromes in less than 2^{n-1} = (diameter/2) time.

We shall now show that there does exist a BTA that accepts the palindromes in O(diameter) time. As a preliminary to this, we show how the cells in the kth layer of a BTA can be made to count modulo 2^k -- i.e., to change state every 2^k time steps. We do this by outputting 1's from the base cells, repeatedly. A non-base cell outputs 1's on alternate times that it receives 1's from its sons. Readily, this implies that the cells in layer k output 1's at time steps that differ by 2^k. We can regard such a cell as initially being in state 0; changing to state 1 when it receives 1's from its sons; changing back to state 0, and outputting 1, the next time it receives 1's from its sons; and so on.

Now suppose that whenever a cell is in state 0, it copies its left son's value (i.e., gray level), and when it is in state 1, it copies its right son's value. Thus a cell just above the base (in layer 1) copies its left and right sons' values alternately; a cell in layer 2 copies the left and right sons of its left son, then the left and right sons of its right son, repeatedly; and so on. By induction, it follows that a son in layer k copies the values of the cells below it in the base, in left to right sequence, at times $k, k+1, \ldots, k+2^k-1$; and this process then repeats (modulo 2^k). In particular, the apex cell copies the entire base, in sequence, starting at time n.

Using a mirror image of this process, we can also get the apex cell to scan the base in right to left sequence. It can thus compare the two scans, point by point, and accept if their first halves match -- i.e., if no mismatch has been found by the time the apex cell enters state 1. The total time required for this is $n+2^{n-1}$ (the scans start at time n, when the values begin to reach the apex), assuming that the processes of copying values and outputting 1's take unit time. This result is nearly optimal, since we saw above that 2^{n-1} is a lower bound.

5.3 (B)PA's and DLP's

A diameter-limited perceptron (DLP) can be defined as operating in the following way:

a) It detects the presence or absence of a given local property in a neighborhood of each point of its input, where the neighborhoods are of some bounded diameter d.

b) It computes a sum of the form $\Sigma a_i p_i$ over the points of its input, where $p_i=1$ or 0 depending on whether or not the property is satisfied at the given point.

c) It compares the sum to a threshold T, and accepts the input iff. the sum is at least T.

We have already indicated (see Section 4.1) that a CA (or a BTA) can detect the presence or absence of a given local property in a fixed-size neighborhood of each of its base cells. Let us therefore assume that this has already been done, and that the cell has stored the value 1 if the property holds, and 0 if it does not (or if the property was not to be computed at that point). We assume that the number of properties to be computed in this way for each base cell is bounded, so that the cell can store a bit pattern of bounded size that indicates which of them are applicable and valid.

Let us further assume that the coefficients a_i in step (b) have bounded precision, so that they too can be stored in the base cells. Thus each base cell can compute its portion of the sum $\Sigma a_i p_i$, since this sum includes only a bounded number of terms, each of bounded size.

Finally, <u>we assume that the threshold T in step (c) has precision that grows at most linearly with the input size</u>. This implies that T can be stored in the base of a BPA as a unary number by storing 1's in a subset of the base cells.

Given all these assumptions, a PA or BTA can simulate the DLP by summing the 1's to compute T, and also summing the a_i's to compute $\Sigma a_i p_i$. The algorithm for summing the a_i's, which are numbers of bounded size, is a straightforward extension of the counting algorithm of Section 4.2; e.g., if the a_i's are in the range [o,r], the cells can sum them by acting as base (r+1) adders. The digits of the sum will thus be output by the apex node at times n,...,2n (where an addition takes unit time). The digits of T will also be output at these times, and if we sum them too in base (r+1), we can compare digits serially to determine whether or not $\Sigma a_i p_i \geq T$.

In summary, we have shown that PA's can simulate a restricted class of diameter-limited perceptrons -- namely, those for which the three underlined assumptions hold. The simulation requires time t_m+2nt_a, where t_m is the time required for the base to compute the p_i's, and t_a is the time for a non-base cell to do a base (r+1) addition. The DLP computes $\Sigma a_i p_i$ and compares it with T "instantaneously", but this requires it to sum a very large number of inputs simultaneously, which is physically unrealistic.

PA's can also do many things that DLP's cannot do. As a simple example, it is known [7] that a DLP cannot determine the parity of the number of 1's (say) in its input. On the other hand, a BTA can do it very simply as follows: The base cells output their values (0 or 1); each non-base cell outputs 0 if its sons' outputs are the same, and 1 if they are different. Readily, a cell's output is just the parity of the number of 1's below it in the base (0 for even, 1 for odd); thus the apex cell's output is the parity for the whole base, and this output is produced in time n. This algorithm can be easily extended to a two-dimensional BPA.

6. Some Other BPA Algorithms

BTA's can also recognize a variety of other input languages; the following are a few examples:

a) Connectedness: In one dimension, the set of strings in $\{0,1\}^*$ in which the 1's are connected is just the regular set $0^*1^*0^*$. Thus a BTA can recognize this set by simulating an FSA, which checks that there is no 10 pattern in the input to the left of a 01 pattern. More generally, a BTA can count connected components of 1's by counting the number of (say) 01 patterns and adding 1 to the sum if there are 1's at the left end of the input. It is an open question whether a BPA can recognize connectedness of the set of 1's in its two-dimensional input, or whether it can count connected components of 1's.

b) Equality and majority. A BPA can easily check whether the number of 1's in its input is equal to, or greater than, the number of 0's, by simultaneously counting both of them and comparing the counts (see Sections 4.2 and 5.3). Less trivially, a BTA can check whether its input contains exactly two connected runs of 1's both of which have the same length. It verifies that there are exactly two runs by simulating an FSA. At the same time, it counts the 1's to the left of the leftmost 10 pattern, by a straightforward extension of the counting algorithm to inhibit counting 1's that lie to the right of a 10.(Specifically: besides transmitting numbers of 1's, each cell also outputs a special signal s that indicates whether the part of the base below it contains the pattern 10. If a cell receives s from its left son, it copies the left son's count of 1's and ignores the right son's count; otherwise, it adds the two counts. Readily, this results in the cell's counting the number of 1's to the left of the leftmost 10 under it in the base.) Concurrently, the BTA counts the 1's to the right of the rightmost 01 pattern, and it compares these two counts, which allows it to determine whether the runs have equal length (or, for that matter, whether a particular one of them has greater length than the other).

c) Closure properties: Union, intersection, and reversal. Trivally, any finite union (or intersection) of BPA languages is a BPA language, since we can construct a BPA that simulates the acceptors of these languages simultaneously, and accepts iff. one (or all) of them accept(s). The reversal of a BTA langauge is a BPA language, since it is accepted by a mirror-image of the given BTA. We shall see in Section 7.2 that the complement of a BPA language is also a BPA language. A concatenation of two BTA languages is a BTA language, provided we concatenate only strings of the same length, since the left and right subtrees of the BTA can then accept the two halves of the base, and the root (apex cell) can accept if its two sons have accepted. It is less easy to see whether concatenations involving strings of different lengths, or numbers of strings that are not powers of 2, can be accepted. This problem is closely related to that of whether BTA languages are closed under translation -- i.e., if L is a BTA language, is there a BTA that accepts $\#^*\sigma\#^*$ (where # is a special "blank" symbol) iff. $\sigma \in L$? Shift invariance will be discussed further in Section 7.3.

d) <u>The Dyck language</u>. We conclude this section by exhibiting a BTA that accepts the set of well-formed parenthesis strings in $O(\log^2 \text{diameter})$ time.

Note first that, given any parenthesis string, we can reduce it to the form $)^i(^j$, where $i \geq 0$ and $j \geq 0$, by repeatedly cancelling adjacent pairs (). Given two strings whose reduced forms are $)^i(^j$ and $)^k(^\ell$, the reduced form of their concatenation is

$$)^{i+k-j}(^{\ell}, \text{ if } j \leq k \; ; \quad)^{i}(^{\ell+j-k}, \text{ if } j \geq k$$

A string is well-formed iff. its reduced form is the null string (i=j=0). Thus a BTA can recognize well-formed parenthesis strings if each cell can compute the reduced form of the parenthesis string in the part of the base below it; the BTA accepts iff. the apex cell's form is null.

Evidently a cell just above the base can compute its reduced form; this is just [i,j] where i=j=0 if the base contains (); i=1,j=1 if the base is)(; i=2,j=0 if it is)); and i=0,j=2 if it is ((. Suppose, then, that each cell transmits its i and j counts to its father, as in the standard counting algorithm, using two separate "channels" to count i's and j's simultaneously. In order for the father to compute its own count, given the [i,j] and [k,ℓ] counts of its sons, the father must first determine whether $j \leq k$ or $j \geq k$. It can do this as it receives the counts for the first time; and it must then receive them a second time in order to do the necessary addition and subtraction ($i+k-j$ or $\ell+j-k$).

The timing of this process is as follows: Each cell receives the inputs from its sons and compares them to determine whether or not $j \leq k$. For a cell in layer h, these inputs have h bits, since the numbers are between 0 and 2^{h-1} inclusive. The next time it receives them, it computes its own (h+1)-bit outputs. It then waits for the inputs to begin again, and computes its outputs again; this process is repeated. Each cell transmits a special flag bit whenever it begins outputting, so that its father can tell when its inputs begin.

Readily, a cell in layer h first begins to receive its h-bit inputs at time $h(h+1)/2$. It may not begin to receive them again immediately, because of the possible waiting times mentioned above; but the delay, if any, is at most h time steps. Thus it can compute and transmit its (h+1)-bit outputs at times separated by not more than $2h+1$. In particular, the apex cell has computed its output by time at most $(n(n+1)/2) + 2n+1 = O(n^2)$, and if this output is zero, it accepts the input string.

7. Limitations

7.1 CA's Can Simulate PA's

PA's are faster than CA's for some tasks, but they are not inherently more powerful. In fact, a CA can simulate a PA, though the simulation is rather inefficient. For simplicity, we demonstrate this in the one-dimensional case, i.e., we show how a one-dimensional CA can simulate a TA.

Since a TA of base length 2^n contains $2^{n+1}-1$ cells, we use for our simulation a CA of length 2^{n+1}; or, equivalently, we use one of length 2^n, and have each cell simulate a pair of adjacent cells in a length 2^{n+1} array. Each CA cell will simulate one TA cell, where the correspondence is defined by a breadth-first scan of the TA; for example, if n=4, the TA cells

```
              1
            2   3
        4   5   6   7
  8  9  10  11  12  13  14  15
```

are mapped into the CA cells

(0),1,2,3,4,5,6,7,8,9,10,11,12,13,14,15.

In this mapping, the cell in the ith position has its brother(s) in positions $i\pm1$, its father in position $\lfloor i/2 \rfloor$, and its sons in positions $2i$ and $2i+1$.

To simulate a single cycle of the TA, each CA cell must have access to the states of its corresponding TA cell's brothers, father, and sons (if they exist). It can immediately access the brothers, since they are adjacent to it in the CA. If the TA cell is at the end of a row, so that it has only one brother (or none, in the case of the apex cell), the CA can know this fact by specially marking those cells whose positions are powers of 2, corresponding to beginning of TA rows; the marks can then inhibit cells 2^i and 2^i-1 from treating each other as brothers. The marking can be done by a simple adaptation of the son-finding algorithm to be described immediately below.

To send information about its state to its father cell, a CA cell sends two signals leftward, one travelling at three times the speed of the other (i.e., one moving at every time step, the other waiting two time steps between moves). When the fast signal reaches the left end of the CA, it bounces back and moves rightward until it meets the *slow signal*, at which point the signals cancel out. It can be readily verified that this meeting takes place first for the signals from cell 1, then for those from cell 2, etc., so that all the cells can send their signals out simultaneously without danger of confusion. (Meetings of slow signals with fast signals that are still moving leftward are ignored.) Moreover, the pair of signals from cell i meets just at position $\lfloor i/2 \rfloor$. Thus the signals can inform cell $\lfloor i/2 \rfloor$ about the state of cell i. For cell 1, the meeting occurs at cell 0, which is not simulating a TA cell, since cell 1 has no father.

Similarly, to send information to its son cells, a CA cell sends a fast signal leftward and a slow signal rightward. The fast signal bounces off the left end and moves rightward until it overtakes the slow signal and they cancel. Again, this happens exactly in sequence for the signals from cells 1,2,..., so that the signals can all be sent simultaneously. Moreover, the signals from cell i meet just at position 2i, so that the signals can inform cell 2i (and its neighbor 2i+1) about the state of cell i. For cells $2^{n-1},\dots,2^n-1$ the slow signal reaches the right end before being overtaken; in this case the signals can be allowed to die, since these TA cells have no sons.

Note that this process takes time $2^{n+1} + 2^n = 3\cdot 2^n$ (the time required for the messages from cell 15 and cell 7 to reach each other). In other words, simulating a single TA time step takes the CA O(diameter) time.

7.2 BPAs' Behavior is Periodic After Polynomial Time

We saw in Section 5.2 that a BTA requires time 2^{n-1} to recognize palindromes, and showed that it can recognize them in essentially that time. For this purpose, we used a construction in which the cells in layer k behave periodically with period 2^k, by changing from one state to another on alternate times that their sons change state. By an immediate generalization of this construction, if a cell changes state every rth time that its sons change state, then the period of the k-th layer cells is r^k, so that the apex cell has period $r^n = 2^{n\log r} = (2^n)^{\log r}$, which is polynomial in the base length.

We shall now show that the behavior of any BPA must be periodic with period polynomial in the base length. To see this, note that the base cells are finite-state machines with constant input; hence their outputs are periodic, say with periods $p_1,\ldots,p_{2^n}$, where each $p_i \le m$ (the number of states). Furthermore, given any cell whose inputs are periodic, say with periods q,r,s,t, its output must also be periodic with period $\le m\cdot\mathrm{LCM}(q,r,s,t)$. Thus any cell in the kth layer has periodic output with period $\le$

$$\begin{aligned} &m\cdot\mathrm{LCM}[m\cdot\mathrm{LCM}[m\cdot\mathrm{LCM}[\ldots]]] \quad \text{(k repetitions)} \\ &= m^k\cdot\mathrm{LCM}[\text{a set of } p_i\text{'s}] \\ &\le m^k\cdot m! \end{aligned}$$

since the p_i's all $\le m$, so their LCM $\le m!$. In particular, the apex cell has period $\le m^n\cdot m! = 2^{n\log m}\cdot m! = m!(2^n)^{\log m}$, which is polynomial in the base diameter.

It follows from these remarks that the complement of a BPA language is always a BPA language. Indeed, given a BPA, M, that accepts the language L, suppose that the cells of M have m states. We can then construct another BPA,M', that simulates M, and that also "counts" (as in the first paragraph above) with apex cell period greater than $m!(2^n)^{\log m}$. When the apex cell completes one period, if the M simulation has not yet accepted its input, M' can accept.

7.3 Shift Invariance

By definition, PA's have bases whose dimensions are powers of 2, so that their inputs must always be of certain sizes. However, we can use them to recognize inputs of arbitrary size by padding the input out (to the next higher power of 2) with a special symbol, say #, that does not otherwise occur. It is clear that this padding does not affect a (B)PA's capability to count local events, to simulate an FSA (in one dimension), to perform linear threshold operations, or to recognize properties such as parity, equality, or majority on the non-# part of its input.

In general, if there exists a PA,M, that accepts an input language L in some standard position (relative to the borders of the base), then we can construct a PA that accepts L in any position, by simply shifting the input to the standard position and then simulating M. This assumes that the standard position is computable by the PA.

The situation is more complicated as regards BPA's,since they cannot shift their input. For languages that are inherently shift-invariant, such as parity, equality, majority, the Dyck language, etc., there is no problem; but it is harder to see how, e.g., a BTA could accept padded palindromes.

7.4 The Nature of BPA Algorithms

The BPA is defined so that information can be transmitted only up the pyramid. If the cells in the base do not have any positional information in the initial configuration, then clearly they can never know their positions in the base. By induction, no cell higher in the BPA can ever have positional information. Thus no cell can know the position of its subpyramid in the BPA and in particular, no cell can know that it is the root of the BPA. Consequently, if a language is accepted by a BPA then every subpyramid must also accept the language. This enforces strict adherence to the concept of local processing,since the computation on a given cell's base must be completed entirely within that cell's subpyramid. This implies that all BPA algorithms must use a balanced divide-and-conquer method which composes solutions to four problems with base size 2^{n-1} by 2^{n-1} into a single solution to the size 2^n by 2^n problem.

8. Concluding Remarks

A class of pyramid cellular acceptors has been defined which improves the potential lower bound time complexity for recognizing two-dimensional patterns to the logarithm of the input diameter. Conventional cellular arrays require time proportional to the diameter for nontrivial recognition tasks. The tradeoff of time for increased hardware is moderate -- a third more processors in the pyramid than in a one-layer cellular array.

It has been shown that many nontrivial recognition problems can indeed be solved in better than diameter time. Those problems have an inherent structure which multiple levels of parallelism can exploit. This can help to delineate a taxonomy of picture languages based on inherent complexity.

Most of the results presented here are described only in the one-dimensional case. A fuller understanding of the applicability of cellular pyramids to the design and analysis of basic image analysis and recognition algorithms will require further work with two-dimensional picture languages.

References

1. A. Rosenfeld and A. C. Kak, Digital Picture Processing, Academic Press, New York, 1976.

2. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974.

3. S. N. Cole, Real-time computation by n-dimensional iterative arrays of finite-state machine, IEEE Trans. on Computers C-18, 1969, 349-365.

4. W. T. Beyer, Recognition of topological invariants by iterative arrays, MAC TR-66, Massachusetts Institute of Technology, October 1969.

5. A. R. Smith, III, Two-dimensional formal languages and pattern recognition by cellular automata, Conference Record IEEE 12th Annual Symposium on Switching and Automata Theory, 1971, 144-152.

6. S. R. Kosaraju, On some open problems in the theory of cellular automata, IEEE Trans. on Computers C-23, 1974, 561-565.

7. M. Minsky and S. Papert, Perceptrons, The MIT Press, Cambridge, Mass., 1969.

8. S. L. Tanimoto and T. Pavlidis, A hierarchical data structure for picture processing, Computer Graphics and Image Processing 4, 1975, 104-119.

9. A. R. Hanson and E. M. Riseman, Preprocessing cones: a computational structure for scene analysis, AD/A0049-64/3GA, University of Massachusetts, Amherst, Mass., Sept. 1974.

10. A. Klinger and C. R. Dyer, Experiments on picture representation using regular decomposition, Computer Graphics and Image Processing 5, 1976, 68-105.

11. A. R. Smith, III, Cellular automata and formal languages, Conference Record IEEE 11th Annual Symposium on Switching and Automata Theory, 1970, 216-224.

12. J. W. Thatcher, Tree automata: an informal survey, in Currents in the Theory of Computing (A. V. Aho, ed.), Prentice-Hall, Englewood Cliffs, N. J., 1973, 143-172.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER 18 AFOSR-TR- 77-0807	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER 9
4. TITLE (and Subtitle) 6 CELLULAR PYRAMIDS FOR IMAGE ANALYSIS.		5. TYPE OF REPORT & PERIOD COVERED Interim technical rept.,
		6. PERFORMING ORG. REPORT NUMBER 14 TR-544
7. AUTHOR(s) 10 Charles R./Dyer Azriel/Rosenfeld	15 ✓AF	8. CONTRACT OR GRANT NUMBER(s) AFOSR-77-3271-77
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Ctr. Univ. of Maryland College Pk., MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 16 61102F 2304/A2 17 A2
11. CONTROLLING OFFICE NAME AND ADDRESS Math.& Info. Sciences, AFOSR/NM Bolling AFB Wash., DC 20332	11	12. REPORT DATE May 1977
		13. NUMBER OF PAGES 45
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) 12 47p.		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Cellular automata
Image analysis
Pattern recognition
Parallel processing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A class of multilayer bounded cellular arrays is defined for which many useful recognition tasks require time proportional to the logarithm of the diameter of the input.

403 018

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 77-0807	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) CELLULAR PYRAMIDS FOR IMAGE ANALYSIS		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER TR-544
7. AUTHOR(s) Charles R. Dyer Azriel Rosenfeld		8. CONTRACT OR GRANT NUMBER(s) AFOSR-77-3271
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Ctr. Univ. of Maryland College Pk., MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Math.& Info. Sciences, AFOSR/NM Bolling AFB Wash., DC 20332		12. REPORT DATE May 1977
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Cellular automata
Image analysis
Pattern recognition
Parallel processing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A class of multilayer bounded cellular arrays is defined for which many useful recognition tasks require time proportional to the logarithm of the diameter of the input.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)